

The ITAPS project has developed a common interface to parallel mesh which it names iMeshP. This interface uses an MPI-like approach to parallel mesh handling, with explicit treatment of message passing and distributed memory representation of a mesh. The iMeshP interface defines a model for partitioning the entities of a mesh among distinct processes. It describes the distribution of and relationships among entities on different processes. In the model, a partition is a high-level description of the distribution of mesh entities. A parallel communication abstraction is used to manage communication among entities and processes in a partition. A partition assigns entities to subsets called parts. The partition maps each part to a process such that each process may have zero, one, or many parts.

As of version 4.0, MOAB implements only a portion of the iMeshP interface. MOAB treats all parallel-relevant data in terms of the existing data model in iMesh. That is, partitions and parts are implemented using entity sets, with tags used to identify them as partition or part sets. Likewise, parallel aspects of the mesh, such as whether an entity is shared among processors or a ghost entity is stored and retrieved in the form of tags. Convenience functions for accessing parallel mesh information are provided by the ParallelComm class in MOAB.

While we will continue to implement the iMeshP interface as time allows, this implementation will probably never be a complete one. Specific differences include:

- **Parts on a process:** iMeshP represents the parallelism in a mesh using the concept of a Part. Mesh sharing, ghosting, and interfaces between parallel aspects is done at the Part level, rather than the process level. In theory, this provides the option of supporting so-called "over-partitioning", where a mesh is partitioned into more Parts than there are available processes. There are two advantages stated for this approach. First, load balancing can be supported based on exchanging whole parts. Second, Partitions can be computed with a maximum number of Parts, and run on lower numbers of Processes, eliminating the need to compute and store multiple Partitions with a mesh. We believe that these advantages are relatively small, especially given the cost of implementing them. Load balancing of whole Parts is clearly not enough to satisfy many applications; indeed, iMeshP provides lower-level mesh migration functions for this very reason. Furthermore, we believe the difficulty of supporting multiple Partitions, in the same base representation of the mesh or in multiple copies, is more an artifact of how many libraries store Partitions, namely as one file per Part. When Partitions and Parts are stored as entity sets with the mesh in a single file, multiple Partitions and Parts are simply another set of metadata in the file. At an implementation level, representing shared mesh on the basis of Parts inserts a level of indirection in determining whether the other shared copies are on-processor or off-processor, when that is really the reason why shared mesh is relevant to applications. So, it is likely MOAB will continue to support only one-Part-per-processor. Applications wanting a finer partition on their data, e.g. to support FETI-type decompositions, can easily implement those capabilities in terms of entity sets.
- **Decomposition and Partitions/Parts?:** Since only Parts and Partitions are used to represent parallel aspects of a mesh, iMeshP must be used to create Partitions and Parts. Thus, there is no way to create or interact with Partitions and Parts unless the iMeshP interface has been instantiated, which for some implementations may require compilation under MPI. In MOAB, Partitions can be based on any covering of the mesh elements being distributed among Parts; examples of these include material groupings, geometric model groupings, or true partition assignments e.g. from Zoltan. Only after an application has loaded mesh in parallel does MOAB consider a particular Partition as the basis for parallel sharing and communication.
- **Knowledge of sharing data:** iMeshP requires only that a Part with a copy of an entity know the part ID and entity handle corresponding to the owner of the entity. That means that if the Part wants to communicate with any other Part sharing the entity, it must first communicate with the owning Part. In MOAB, if an entity is shared with another Part/process, the handles for the entity on all sharing Parts/processes are

known. This allows direct communication between all Parts/processes sharing an entity. In practice, this results in fewer round-trip messages, decreasing communication costs in parallel applications.

These differences are all that are known currently, though there are discussions underway regarding iMeshP that could result in further differences. This wiki page will be updated once those discussions are concluded.